Complexity notes

Priit Järv, TUT 2015

How long does this function execute?

```
def sum(list):
   sum = 0
   for elem in list:
      sum += elem
   return sum
```

(this function computes $\sum a_i$ for $\{a_1, \ldots, a_n\}$)

How long does this function execute?

```
def sum(list):
    sum = 0
    for elem in list:
        sum += elem
    return sum
```

Count the number of code lines:

2n + 2 if the length of list is n

Complexity of algorithms

Estimate of execution time $T(n) = 2n+2$ for input n

The big-O notation (asymptotic complexity):

Upper bound of T(n) is O(f(n))

if $T(n) \leq kf(n)$ for some k and $n>n_o$

In this case, O(n) because we can choose (for example) k=3 and $n_o=1$. Then $2n+2 \leq 3n$ for $n>1$.

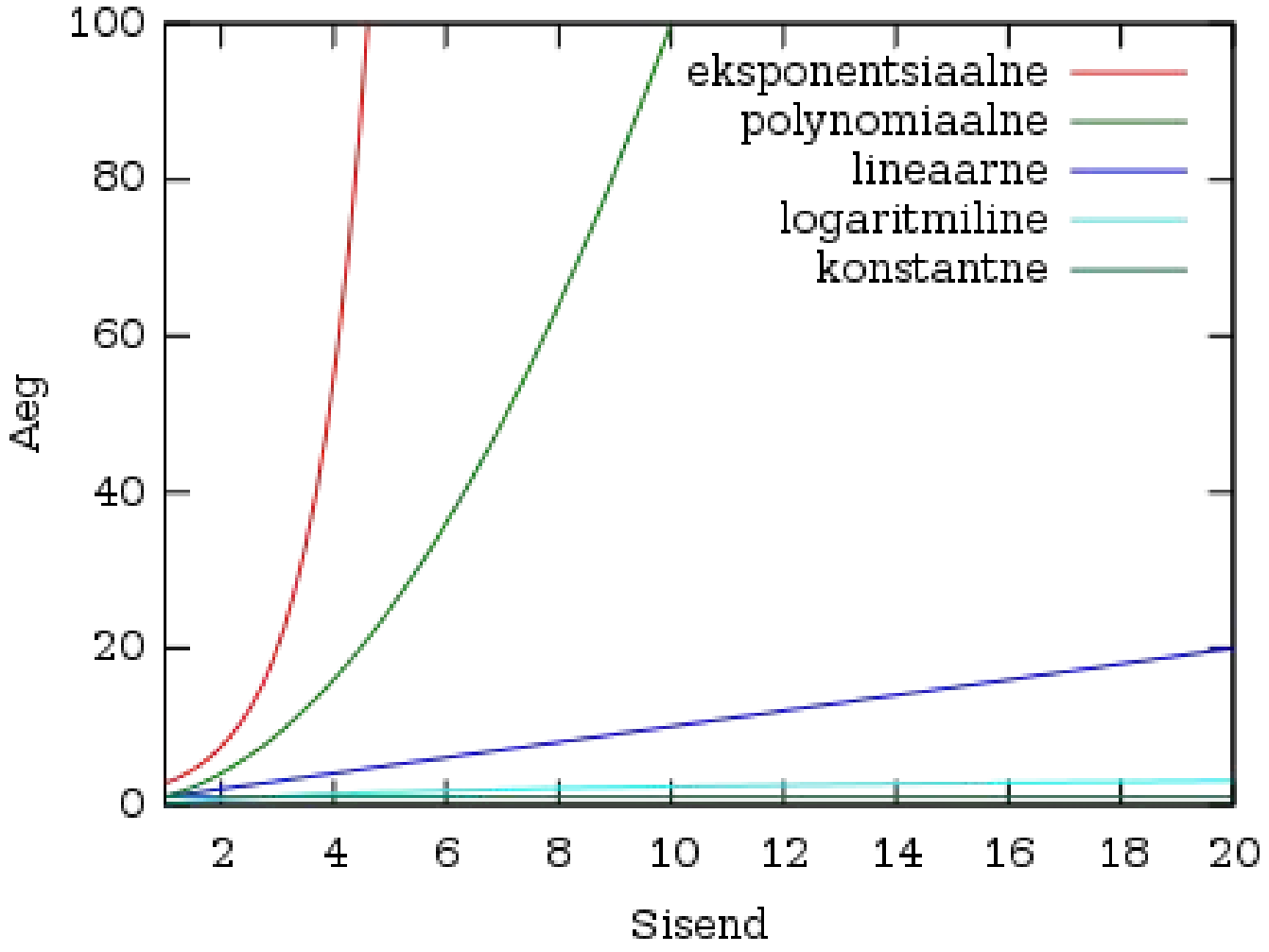
Complexity of algorithms

The practical usefulness of the big-O notation:

	problem size increase	time increase
$O(n)$	5x	5x
$O(n^2)$	5x	25x
$O(2^n)$	5x	A LOT*

*- for example, about 10^{12} times if increasing from 10 to 50

Common complexity classes



Polynomial time

$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_o$ is n-th order polynomial

These are also polynomials:

$2x$, 6, x^3

$O(n^3)$ - polynomial complexity

the algorithm takes input size cubed time to run.

Common simplifications

k - constant

$O(kn) = O(n)$

$O(g(n)) = O(n^k)$ if g(n) is *k*-th order polynomial

Complexity of problems

There are two main classes: P and NP

P - solvable in polynomial time

NP - ALSO solvable in polynomial time

(on a Non-deterministic Turing machine)

NP-complete - hardest problems in NP

Complexity of problems

Practical implications:

- No polynomial time algorithms for NP-complete problems.
- A new problem is also NP-complete if we can easily convert a known NP-complete problem to the new problem.